***Faculté des sciences - Université de Montpellier - Master Informatique***

# PROJET SCIENCES DES DONNÉES

## Classification d'assertions selon leur valeurs de véracité (fact-checking)

Réalisé par :

*Elbachir REHHALI : 21812980*
*Mohamed HASSAN IBRAHIM : 21914461*
*Karim DAHDOUH : 21914506*
*Thierno BARRY : 21914683*

Année anniversaire 2019-2020

## Importation des Modules utils pour faire de l'apprentissage supervisé

Roles de quelques modules :

- Pandas : Pandas est une bibliothèque python permettant la manipulation et l'analyse des données. Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles.
- Numpy : destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.
- NLTK: aussi appelé Natural language Toolkit est une bibliothèque python pour le traitement automatique des langues.
- scikit learn : c'est une bibliothèque libre Python destinée à l'apprentissage automatique.C'est donc elle nous ferons l'apprentissage.
- etc.

In [125]:

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

import pickle
import pandas as pd
pd.options.mode.chained_assignment = None # pour désactiver warning
import numpy as np
from nltk.tokenize import word_tokenize
import nltk
from nltk.stem import WordNetLemmatizer
stemmer = WordNetLemmatizer()
#nltk.download()
import unicodedata
import re
import inflect
from collections import Counter
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from time import time

import matplotlib.pyplot as plot
import seaborn as sns
import altair as alt


from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler

from sklearn.preprocessing import FunctionTransformer
from sklearn.pipeline import FeatureUnion, Pipeline
```

## Importation du dataset avec les colonnes qui nous interessent.

In [126]:

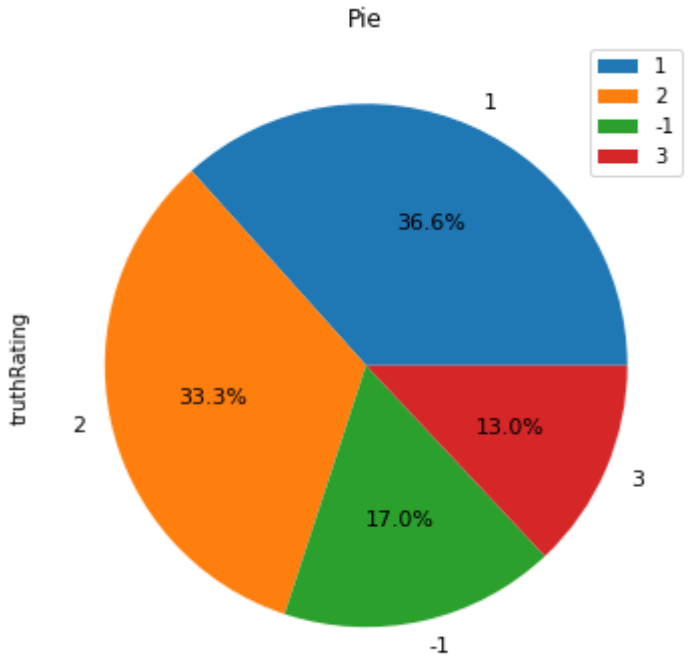
```
url="claimskg_result.csv"
df = pd.read_csv(url, usecols=["text", "headline", "truthRating", "source", "keywor
df_part2 = df  # utiliser dans deuxieme partie de notebook
# Dimension de notre dataFrame
print("shape: ", df.shape)
display(df.head())
```

```
shape:  (2000, 6)
```

| | text | author | headline | keywords | source | truthRating |
|---|---|---|---|---|---|---|
| **0** | Malia Obama cashed a $1.2 million tax refund c... | Unknown | Did Malia Obama Cash a $1.2 Million Check? | NaN | truthorfiction | -1 |
| **1** | High diver is saved from jumping into a draine... | Unknown | High Diver Saved By Cross | ASP Article | snopes | -1 |
| **2** | 'And the revenue generated by drilling off Vir... | Jim Moran | Moran says drilling off Virginia's coast will ... | Energy,State Finances | politifact | 2 |
| **3** | Health insurance companies pay CEOs $24 millio... | Health Care for America Now | Health care advocacy group blasts insurers for... | Corporations,Health Care | politifact | 2 |
| **4** | Ted Cruz said that veterans should start selli... | Unknown | Ted Cruz: Vets Should Sell Cookies for Funding... | ASP Article, Not Necessarily The News | snopes | 1 |

In [127]:

```
df['truthRating'].value_counts().plot(kind='pie',
figsize=(6,6),
title='Pie',
fontsize=11,
legend=True,
autopct='%1.1f%%')

plot.show()
```




# Partie 1: Dataset (TRUE & FALSE)

Comme l'indique le sujet, nous devons faire deux classifications binaires qui sont: -TRUE vs FALSE -TRUE & FALSE vs MIXTURE Nous avons donc commencé par TRUE vs FALSE

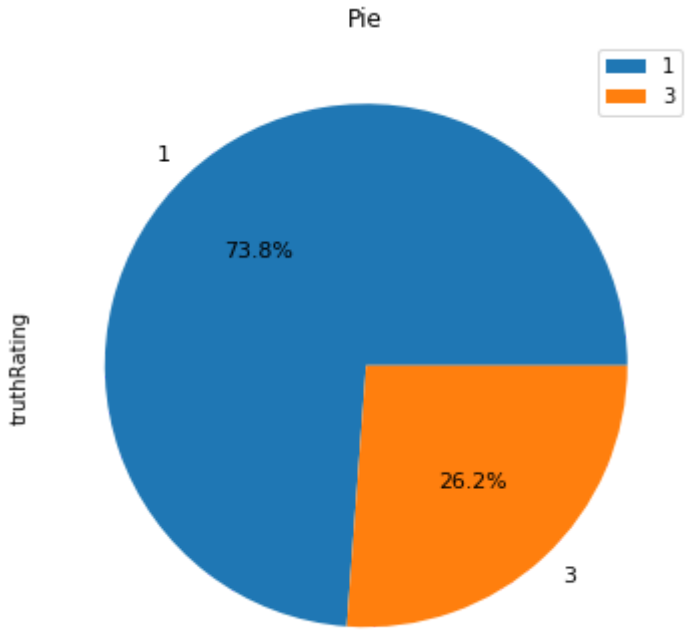
In [128]:

```
#filter dataframe pour récuperer que les TRUE & FALSE dont on a besoin.
df = df[(df['truthRating'] == 3) | (df['truthRating'] == 1)]
df_v2 = df
# La nouvelle taille de notre DataFrame
print(df.shape)
# Affihage du nouveau dataframe obtenu après le filtrage.
display(df)
```

```
(993, 6)
```

In [129]:

```
df['truthRating'].value_counts().plot(kind='pie',
figsize=(6,6),
title='Pie',
fontsize=11,
legend=True,
autopct='%1.1f%%')

plot.show()
```




le 1 represente une assertion fause marquée(FALSE) et le 3 une assertion vraie(TRUE) nous constatons donc une grande différence entre les deux classes.

# Version 1: sans appliquer les prétraitements des données

## Préparation et Test de notre modèle avec la feature principale text

In [130]:

```
vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(df.text)
#Specification des variables à prédire et de la classe X et y
X = vectors.toarray()

y = df.truthRating
```

## Création des datasets d'apprentissage et de test

On est en apprentissage supervisé et il est récommendé de faire de l'apprentissage sur 70% de nos données et ensuite tester notre modèle sur les 30%.

In [131]:

```
from sklearn.model_selection import train_test_split

validation_size=0.7 #30% du jeu de données pour le test

testsize= 1-validation_size
seed=30
X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)
```

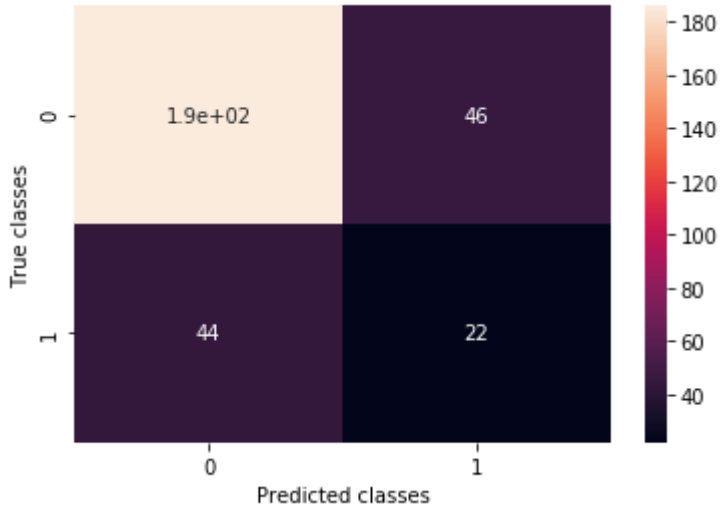
## Classifieur Gaussian Naive Bayes & Sa matrice de confusion

In [132]:

```
clf = GaussianNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('F1 score(Accuracy): ', f1_score(y_test, y_pred, pos_label='positive', averag

## classification report
conf = confusion_matrix(y_test, y_pred)
print ('\n matrice de confusion \n',conf)

print ('\n',classification_report(y_test,
                                  y_pred))

## classification chart
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True)
plot.ylabel('True classes')
plot.xlabel('Predicted classes')
plot.show()
```

```
/home/karim/env/lib/python3.6/site-packages/sklearn/metrics/_classific
ation.py:1321: UserWarning: Note that pos_label (set to 'positive') is
ignored when average != 'binary' (got 'micro'). You may use labels=[po
s_label] to specify a single positive class.
  % (pos_label, average), UserWarning)
```

```
F1 score(Accuracy):  0.697986577181208

 matrice de confusion
 [[186  46]
 [ 44  22]]

              precision    recall  f1-score   support

           1       0.81      0.80      0.81       232
           3       0.32      0.33      0.33        66

    accuracy                           0.70       298
   macro avg       0.57      0.57      0.57       298
weighted avg       0.70      0.70      0.70       298
```




## Préparation et Test de notre modèle avec la feature principale text et d'autres metadata toujours sans prétraitement

## Variables d'appprentissage et de prédiction

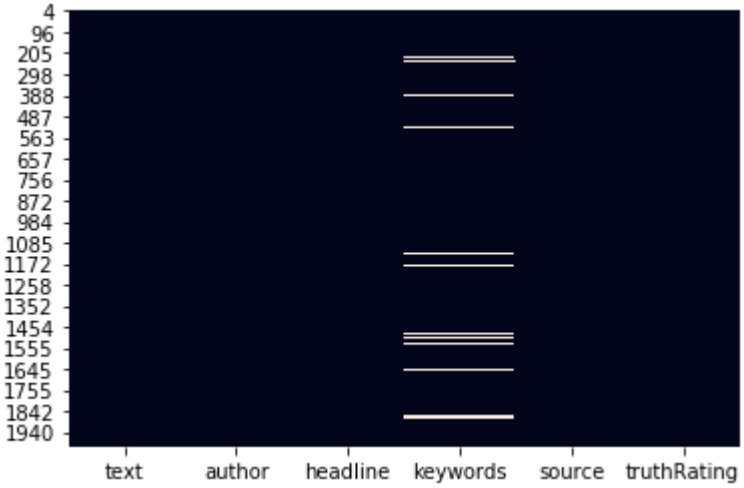
On définit les variables d'apprentissage et la variable à prédire. truthRating est la variable à prédire. et les autres à savoir le texte et les metadata sont les variables utilisées pour faire l'apprentissage en vue d'arriver à prédire la véracité d'un article.

### *Chercher et enlever les valeurs manquantes*

In [133]:

```
sns.heatmap(df.isnull(), cbar=False)
count_row_orig = df.shape[0]
print ('Suppression des lignes pour lesquelles au moins un élément est manquant \n'
df.dropna(inplace=True)

print("nombre de ligne supprimer = ", count_row_orig - df.shape[0])
plot.show()
```

```
Suppression des lignes pour lesquelles au moins un élément est manquan
t

nombre de ligne supprimer =  68
```




## Choix des variables pour l'apprentissage ainsi que la variable à prédire

In [134]:

```
print(df.shape)
array = df.values

#Les variables d'apprentissage ("text", "headline", "source", "keywords", "author")
X = array[:,  [0,1,2,3,4]]
# La variable à prédire("truthRating").
y = array[:, 5]
y=y.astype('int')
```

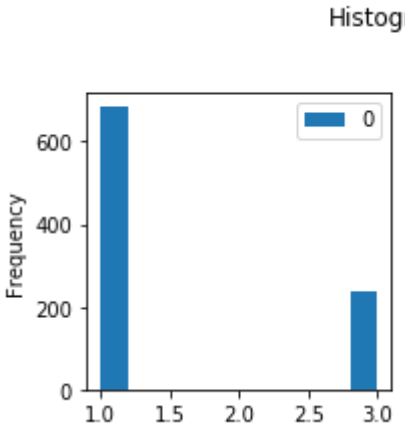
```
(925, 6)
```

## Visualisation sur les variables à prédire (True et False) :

C'est avec la visualisation, nous pouvons avoir une meilleure compréhension de nos données et ceci nous permettra de faire le choix pour les prochaines étapes.

In [135]:

```
pd.DataFrame(data=y).plot(kind='hist', subplots=True,
 layout=(2,2), figsize=(6,6),
 sharex=False, title='Histogramme')
plot.show()
```




## Mise en place du pipline Et Application de la vectorisation.

In [93]:

```
df2 = pd.DataFrame(data=X, columns=["text", "headline", "source", "keywords", "auth

#print(df2.shape)
# Mise en place d'un pipeline des données et appliquer TfidfVectorizer (pour transf

transformer = FeatureUnion([
                ('text_tfidf',
                  Pipeline([('extract_field',
                             FunctionTransformer(lambda x: x['text'],
                                                 validate=False)),
                            ('tfidf',
                             TfidfVectorizer(ngram_range=(1, 2)))])),
                ('headline_tfidf',
                  Pipeline([('extract_field',
                             FunctionTransformer(lambda x:  x["headline"],
                                                 validate=False)),
                            ('tfidf',
                             TfidfVectorizer(ngram_range=(1, 2)))])),
                ('keywords_tfidf',
                  Pipeline([('extract_field',
                             FunctionTransformer(lambda x:  x["keywords"],
                                                 validate=False)),
                            ('tfidf',
                             TfidfVectorizer(ngram_range=(1, 2)))])),
                ('author_tfidf',
                  Pipeline([('extract_field',
                             FunctionTransformer(lambda x:  x["author"],
                                                 validate=False)),
                            ('tfidf',
                             TfidfVectorizer(ngram_range=(1, 2)))])),
                ('source_tfidf',
                  Pipeline([('extract_field',
                             FunctionTransformer(lambda x:  x["source"],
                                                 validate=False)),
                            ('tfidf',
                             TfidfVectorizer())]))
                ])

transformer.fit(df2)


text_vocab = transformer.transformer_list[0][1].steps[1][1].get_feature_names()
headline_vocab = transformer.transformer_list[1][1].steps[1][1].get_feature_names()
keywords_vocab = transformer.transformer_list[2][1].steps[1][1].get_feature_names()
author_vocab = transformer.transformer_list[3][1].steps[1][1].get_feature_names()
source_vocab = transformer.transformer_list[4][1].steps[1][1].get_feature_names()

#print(source_vocab)


vocab = text_vocab + headline_vocab + keywords_vocab + author_vocab  + source_vocab


X = pd.DataFrame(
    data=transformer.transform(df2).toarray(),
    columns=vocab
)
#Cinq premières lines
display(X.head())
```

| | 000 | 000 000 | 000 abortions | 000 americans | 000 and | 000 babies | 000 during | 000 feet | 000 for | 000 from | ... | yousef | zealand |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **1** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **2** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **3** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |

5 rows × 23120 columns

## Création des datasets d'apprentissage et de test

On est en apprentissage supervisé et il est récommendé de faire de l'apprentissage sur 70% de nos données et ensuite tester notre modèle sur les 30%.

In [94]:

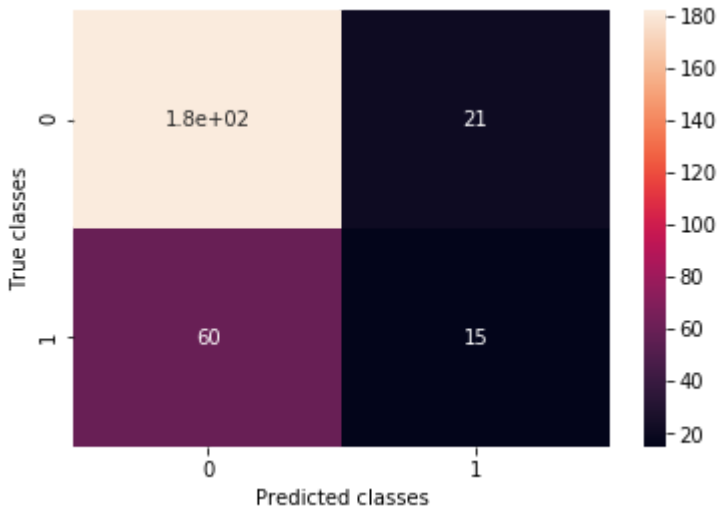
```
from sklearn.model_selection import train_test_split
#30% du jeu de données pour le test
validation_size=0.7
#70% du jeu de données pour l'apprentissage
testsize= 1-validation_size
seed=30
X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)
```

## Classifieur Gaussian Naive Bayes & Sa matrice de confusion

In [95]:

```
clf = GaussianNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('F1 score(Accuracy): ', f1_score(y_test, y_pred, average='micro'))

## classification report
conf = confusion_matrix(y_test, y_pred)
print ('\n matrice de confusion \n',conf)

print ('\n',classification_report(y_test,
                                  y_pred))

## classification chart
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True)
plot.ylabel('True classes')
plot.xlabel('Predicted classes')
plot.show()
```

```
F1 score(Accuracy):  0.7086330935251799

 matrice de confusion
 [[182  21]
 [ 60  15]]

              precision    recall  f1-score   support

           1       0.75      0.90      0.82       203
           3       0.42      0.20      0.27        75

    accuracy                           0.71       278
   macro avg       0.58      0.55      0.54       278
weighted avg       0.66      0.71      0.67       278
```

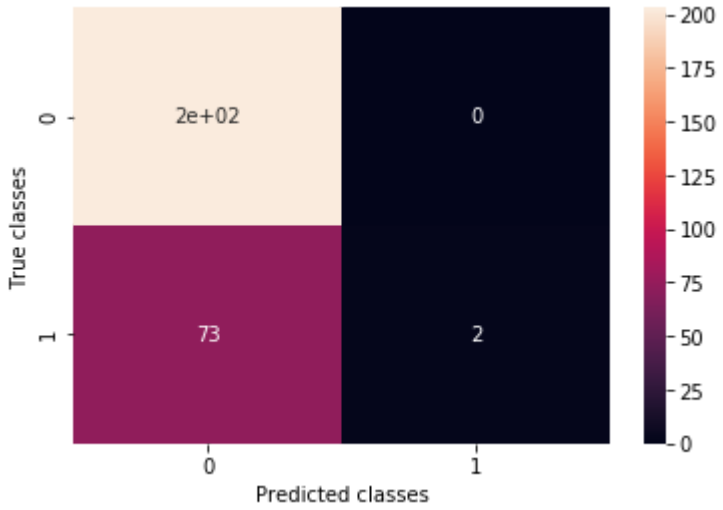



## Classifieur RandomForest & Sa matrice de confusion

In [96]:

```
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('F1 score(Accuracy): ', f1_score(y_test, y_pred, average='micro'))

## classification report
conf = confusion_matrix(y_test, y_pred)
print ('\n matrice de confusion \n',conf)

print ('\n',classification_report(y_test,
                                  y_pred))

## classification chart
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True)
plot.ylabel('True classes')
plot.xlabel('Predicted classes')
plot.show()
```

```
F1 score(Accuracy):  0.737410071942446

 matrice de confusion
 [[203   0]
 [ 73   2]]

              precision    recall  f1-score   support

           1       0.74      1.00      0.85       203
           3       1.00      0.03      0.05        75

    accuracy                           0.74       278
   macro avg       0.87      0.51      0.45       278
weighted avg       0.81      0.74      0.63       278
```



# Version 2: appliquer les prétraitements des données

## Ingénierie des données : Opération de prétraitement de nos données

- supprimer les caractères non Ascii
- uniformiser le texte en mettant tous les mots en minuscule
- Supprimer les ponctuations
- remplacer les nombres par des mots
- enlever les stopwords.
- supprimer les caractères spéciaux
- lemmatisation

In [97]:

```
from sklearn.feature_extraction.text import CountVectorizer

print(df.shape)
array = df.values

#Les variables d'apprentissage
X = array[:,  [0,1,2,3,4]]
# La variable à prédire.
y = array[:, 5]
y=y.astype('int')
```

```
(925, 6)
```

### UnderSampling & OverSampling

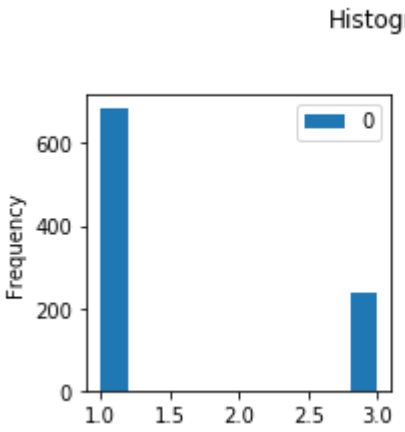
Le Oversampling(Suréchantillonage) et le UnderSampling(sous-échantillonage) sont utilisées pour ajuster la distribution des classes d'un ensemble de données. Dans notre cas, nous voyons bien à partir de la visualisation qu'il y a une grande différence entre les classses TRUE & FALSE.

### OverSampling (la meilleure solution dans notre cas)

#### Avant d'appliquer l'Oversampling

In [152]:

```
pd.DataFrame(data=y).plot(kind='hist', subplots=True,
 layout=(2,2), figsize=(6,6),
 sharex=False, title='Histogramme')
plot.show()
```

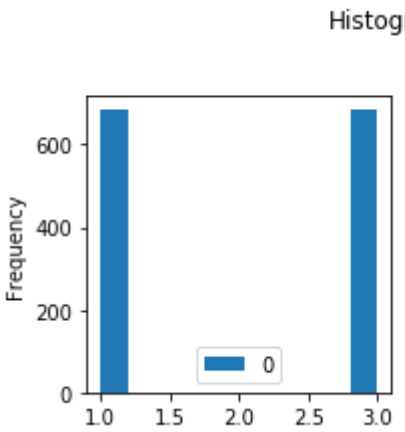



**Après avoir appliquer l'Oversampling**

In [155]:

```
print('y ==> Original dataset shape %s' % Counter(y))
ros = RandomOverSampler(random_state=42)
X_res, y_res = ros.fit_resample(X, y)
print('y ==> over sampling dataset shape %s' % Counter(y_res))

X = X_res
y = y_res

pd.DataFrame(data=y).plot(kind='hist', subplots=True,
 layout=(2,2), figsize=(6,6),
 sharex=False, title='Histogramme')
plot.show()
```

```
y ==> Original dataset shape Counter({1: 687, 3: 238})
y ==> over sampling dataset shape Counter({1: 687, 3: 687})
```




***Liste des fonctions de prétraitements***

In [99]:

```
#Fonction de suppression des caractères non ASCII
def remove_non_ascii(words):
    new_words = []
    for word in words:
        new_word = unicodedata.normalize('NFKD', word).encode('ascii', 'ignore').de
        new_words.append(new_word)
    return new_words

#uniformiser le texte en mettant tous les mots en minuscule
def to_lowercase(words):
    new_words = []
    for word in words:
        new_word = word.lower()
        new_words.append(new_word)
    return new_words

#Supprimer les ponctuations
def remove_punctuation(words):
    new_words = []
    for word in words:
        new_word = re.sub(r'[^\w\s]', '', word)
        if new_word != '':
            new_words.append(new_word)
    return new_words

#remplacer les nombres par des mots
def replace_numbers(words):
    p = inflect.engine()
    new_words = []
    for word in words:
        if word.isdigit():
            new_word = p.number_to_words(word)
            new_words.append(new_word)
        else:
            new_words.append(word)
    return new_words
#enlever les stopwords.
def remove_stopwords(words):
    new_words = []
    for word in words:
        if word not in stopwords.words('english'):
            new_words.append(word)
    return new_words



# Supprimer les caractères spéciaux
def remove_special_caracters(words):
    new_words = []
    for word in words:
        #new_word = re.sub(r'\W', ' ', word)
        new_word = re.sub("[^A-Za-z0-9]", " ", word)
        if new_word != '':
            new_words.append(new_word)
    return new_words

# Substituting multiple spaces with single space
def substitute_multiplecaracters_bysingle(words):
    new_words = []
```

```
    for word in words:
        new_word = re.sub(r'\s+', ' ', word, flags=re.I)
        if new_word != '':
            new_words.append(new_word)
    return new_words

# lemmatisation
def word_lemmatization(words):
    new_words = []
    for word in words:
        new_word = stemmer.lemmatize(word)
        new_words.append(new_word)
    return new_words



# regroupement de toutes les fonctions dans une seule.
def normalize(words):
    print("===========ASCI pré-traitements==========")
    words = remove_non_ascii(words)
    print(words)
    print("=========================================================================
    print("==================== pré-traitements : mettre MINUSCULES pré-traitements
    words = to_lowercase(words)
    print(words)
    print("=========================================================================
    print("=============== pré-traitements : supprimer PONCTUATION pré-traitements
    words = remove_punctuation(words)
    print(words)
    print("=========================================================================
    print("================ pré-traitements : supprimer STOPWORDS pré-traitements =
    words = remove_stopwords(words)
    print(words)
    print("=========================================================================
    print("=========== pré-traitements : supprimer CARACTÈRES SPÉCIAUX pré-traiteme
    words = remove_special_caracters(words)
    print(words)
    print("=========================================================================
    print("============  pré-traitements : substituer plusieurs espaces par un seul
    words = substitute_multiplecaracters_bysingle(words)
    print(words)
    print("=========================================================================
    print("================  pré-traitements : replacer les nombres par des mots
    words = replace_numbers(words)
    print(words)
    print("=========================================================================
    #words = word_lemmatization(words)
    return words

def clean_text(text):
    tokens = word_tokenize(text)
    tokens=normalize(tokens)
    text="".join([" "+i for i in tokens]).strip()
    return text
```

## Notoyage des assertions

Nous allons maintenant appliquer nos fonctions de prétraiement à notre texte.

In [100]:

```
def clean_assertions (data):
    for i in range(len(data)):
        #print (i)
        data[i]=clean_text(data[i])
    return data
```

In [101]:

```
# On parcours les assertions et on les nettoie
for i in range(X.shape[0]):
    X[i] = clean_assertions(X[i])
```

```
==========ASCI pré-traitements==========
['Ted', 'Cruz', 'said', 'that', 'veterans', 'should', 'start', 'sell
ing', 'cookies', 'in', 'order', 'to', 'raise', 'funds', '.']
========================================================================
==================
==================== pré-traitements : mettre MINUSCULES pré-traitem
ents==============
['ted', 'cruz', 'said', 'that', 'veterans', 'should', 'start', 'sell
ing', 'cookies', 'in', 'order', 'to', 'raise', 'funds', '.']
========================================================================
==================
=============== pré-traitements : supprimer PONCTUATION pré-traiteme
nts ===============
['ted', 'cruz', 'said', 'that', 'veterans', 'should', 'start', 'sell
ing', 'cookies', 'in', 'order', 'to', 'raise', 'funds']
========================================================================
==================
=============== pré-traitements : supprimer STOPWORDS pré-traitemen
ts ================
['ted', 'cruz', 'said', 'veterans', 'start', 'selling', 'cookies'
```

## Mise en place du pipline Et Application de la vectorisation.

In [33]:

```
df2 = pd.DataFrame(data=X, columns=["text", "headline", "source", "keywords", "auth

#print(df2.shape)
# Mise en place d'un pipeline des données et appliquer TfidfVectorizer (pour transf
transformer = FeatureUnion([
                ('text_tfidf',
                  Pipeline([('extract_field',
                             FunctionTransformer(lambda x: x['text'],
                                                 validate=False)),
                            ('tfidf',
                              TfidfVectorizer(ngram_range=(1, 2)))])),
                ('headline_tfidf',
                  Pipeline([('extract_field',
                             FunctionTransformer(lambda x:  x["headline"],
                                                 validate=False)),
                            ('tfidf',
                              TfidfVectorizer(ngram_range=(1, 2)))])),
                ('keywords_tfidf',
                  Pipeline([('extract_field',
                             FunctionTransformer(lambda x:  x["keywords"],
                                                 validate=False)),
                            ('tfidf',
                              TfidfVectorizer(ngram_range=(1, 2)))])),
                ('author_tfidf',
                  Pipeline([('extract_field',
                             FunctionTransformer(lambda x:  x["author"],
                                                 validate=False)),
                            ('tfidf',
                              TfidfVectorizer(ngram_range=(1, 2)))])),
                ('source_tfidf',
                  Pipeline([('extract_field',
                             FunctionTransformer(lambda x:  x["source"],
                                                 validate=False)),
                            ('tfidf',
                              TfidfVectorizer())]))
                ])

transformer.fit(df2)


text_vocab = transformer.transformer_list[0][1].steps[1][1].get_feature_names()
headline_vocab = transformer.transformer_list[1][1].steps[1][1].get_feature_names()
keywords_vocab = transformer.transformer_list[2][1].steps[1][1].get_feature_names()
author_vocab = transformer.transformer_list[3][1].steps[1][1].get_feature_names()
source_vocab = transformer.transformer_list[4][1].steps[1][1].get_feature_names()

#print(source_vocab)


vocab = text_vocab + headline_vocab + keywords_vocab + author_vocab  + source_vocab


X = pd.DataFrame(
    data=transformer.transform(df2).toarray(),
    columns=vocab
)
#Cinq premières lines
display(X.head())
```

| | 101st | 101st birthday | 15th | 15th anniversary | 15th largest | 15yearold | 15yearold boy | 15yearold single | 18cabin | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **1** | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **2** | 0.0 | 0.0 | 0.196194 | 0.216076 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **3** | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| **4** | 0.0 | 0.0 | 0.000000 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |

5 rows × 19434 columns

# Création des datasets d'apprentissage et de test

On est en apprentissage supervisé et il est récommendé de faire de l'apprentissage sur 70% de nos données et ensuite tester notre modèle sur les 30%.

In [34]:

```
from sklearn.model_selection import train_test_split
#30% du jeu de données pour le test
validation_size=0.7
#70% du jeu de données pour l'apprentissage
testsize= 1-validation_size
seed=30
X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)
```

# Début de la classification

Nous allons maintenant appliquer différents classifieurs sur nos données Pour chaque classifieur :

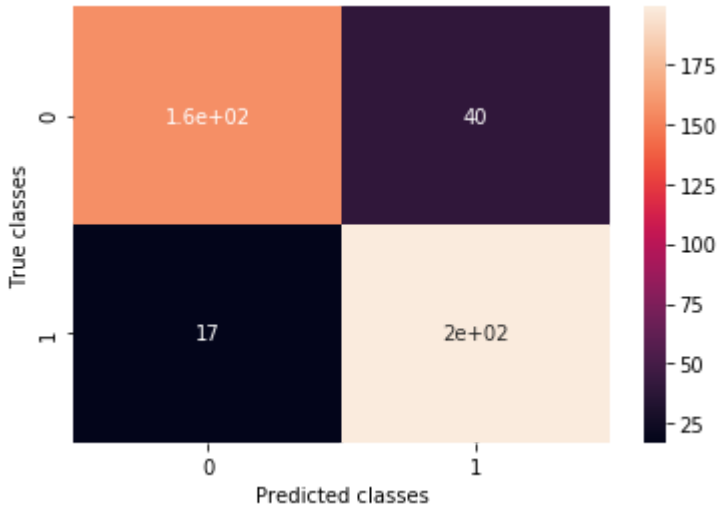
- On test notre modèle appris
- on affiche Accuracy en utilisant la fonction accuracy_score()
- on affiche sa matrice de confusion en utilisant la fonction confusion_matrix()
- on affiche les scores des métriques Precision, Recall, F1-Score et Support en utilisant la fonction classification_report()

## Classifieur Gaussian Naive Bayes & Sa matrice de confusion

In [23]:

```
clf = GaussianNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('F1 score(Accuracy): ', f1_score(y_test, y_pred, average='micro'))

## classification report
conf = confusion_matrix(y_test, y_pred)
print ('\n matrice de confusion \n',conf)

print ('\n',classification_report(y_test,
                                  y_pred))

## classification chart
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True)
plot.ylabel('True classes')
plot.xlabel('Predicted classes')
plot.show()
```

```
F1 score(Accuracy):  0.8619854721549637

 matrice de confusion
 [[157  40]
 [ 17 199]]

              precision    recall  f1-score   support

           1       0.90      0.80      0.85       197
           3       0.83      0.92      0.87       216

    accuracy                           0.86       413
   macro avg       0.87      0.86      0.86       413
weighted avg       0.87      0.86      0.86       413
```



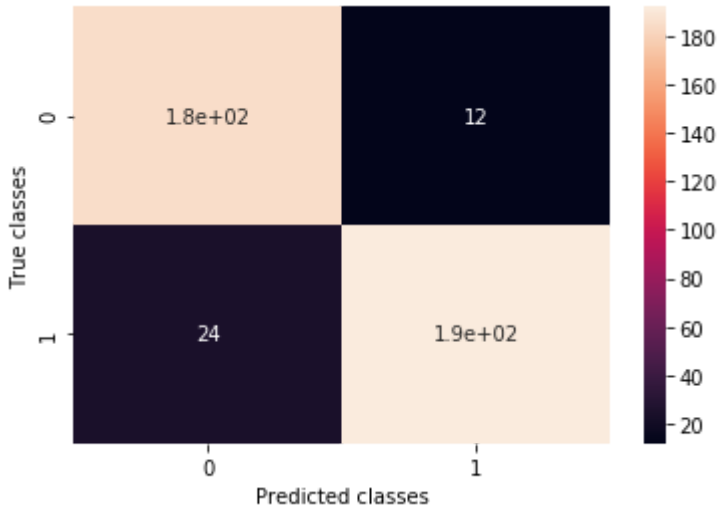
# Classifieur RandomForest & Sa matrice de confusion

In [24]:

```
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('F1 score(Accuracy): ', f1_score(y_test, y_pred, pos_label='positive', averag

## classification report
conf = confusion_matrix(y_test, y_pred)
print ('\n matrice de confusion \n',conf)

print ('\n',classification_report(y_test,
                                  y_pred))

## classification chart
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True)
plot.ylabel('True classes')
plot.xlabel('Predicted classes')
plot.show()
```

```
/home/karim/env/lib/python3.6/site-packages/sklearn/metrics/_classific
ation.py:1321: UserWarning: Note that pos_label (set to 'positive') is
ignored when average != 'binary' (got 'micro'). You may use labels=[po
s_label] to specify a single positive class.
  % (pos_label, average), UserWarning)
```

```
F1 score(Accuracy):  0.9128329297820823

 matrice de confusion
 [[185  12]
 [ 24 192]]

              precision    recall  f1-score   support

           1       0.89      0.94      0.91       197
           3       0.94      0.89      0.91       216

    accuracy                           0.91       413
   macro avg       0.91      0.91      0.91       413
weighted avg       0.91      0.91      0.91       413
```



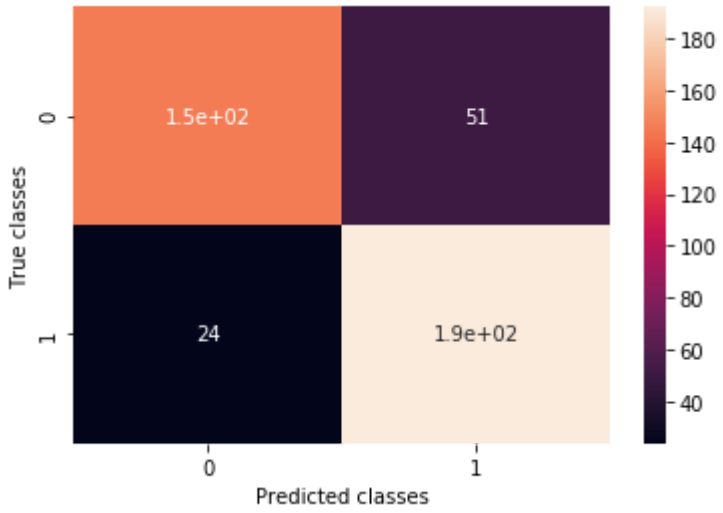
# Classifieur DecisionTree & Sa matrice de confusion

In [25]:

```
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('F1 score(Accuracy): ', f1_score(y_test, y_pred, pos_label='positive', averag

## classification report
conf = confusion_matrix(y_test, y_pred)
print ('\n matrice de confusion \n',conf)

print ('\n',classification_report(y_test,
                                  y_pred))

## classification chart
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True)
plot.ylabel('True classes')
plot.xlabel('Predicted classes')
plot.show()
```

```
/home/karim/env/lib/python3.6/site-packages/sklearn/metrics/_classific
ation.py:1321: UserWarning: Note that pos_label (set to 'positive') is
ignored when average != 'binary' (got 'micro'). You may use labels=[po
s_label] to specify a single positive class.
  % (pos_label, average), UserWarning)
```

```
F1 score(Accuracy):  0.8184019370460048

 matrice de confusion
 [[146  51]
 [ 24 192]]

              precision    recall  f1-score   support

           1       0.86      0.74      0.80       197
           3       0.79      0.89      0.84       216

    accuracy                           0.82       413
   macro avg       0.82      0.82      0.82       413
weighted avg       0.82      0.82      0.82       413
```

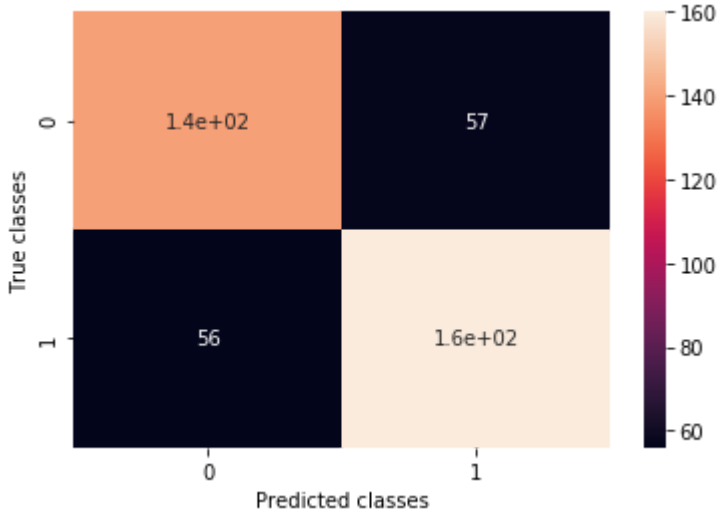


# Classifieur Nearest Neighbors & Sa matrice de confusion

In [26]:

```
clf = KNeighborsClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('F1 score(Accuracy): ', f1_score(y_test, y_pred, average='micro'))

## classification report
conf = confusion_matrix(y_test, y_pred)
print ('\n matrice de confusion \n',conf)

print ('\n',classification_report(y_test,
                                  y_pred))

## classification chart
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True)
plot.ylabel('True classes')
plot.xlabel('Predicted classes')
plot.show()
```

```
F1 score(Accuracy):  0.7263922518159807

 matrice de confusion
 [[140  57]
 [ 56 160]]

              precision    recall  f1-score   support

           1       0.71      0.71      0.71       197
           3       0.74      0.74      0.74       216

    accuracy                           0.73       413
   macro avg       0.73      0.73      0.73       413
weighted avg       0.73      0.73      0.73       413
```

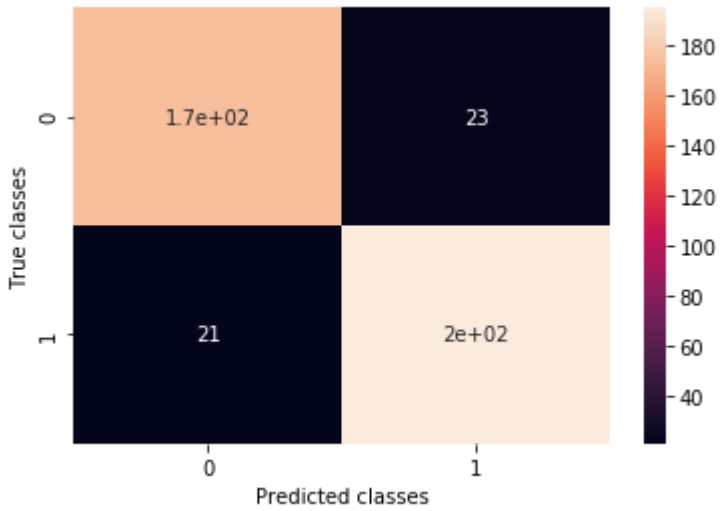



# Classifieur Linear SVC & Sa matrice de confusion

In [27]:

```
clf = LinearSVC()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('F1 score(Accuracy): ', f1_score(y_test, y_pred, average='micro'))

## classification report
conf = confusion_matrix(y_test, y_pred)
print ('\n matrice de confusion \n',conf)

print ('\n',classification_report(y_test,
                                  y_pred))

## classification chart
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True)
plot.ylabel('True classes')
plot.xlabel('Predicted classes')
plot.show()
```

```
F1 score(Accuracy):  0.8934624697336562

 matrice de confusion
 [[174  23]
 [ 21 195]]

              precision    recall  f1-score   support

           1       0.89      0.88      0.89       197
           3       0.89      0.90      0.90       216

    accuracy                           0.89       413
   macro avg       0.89      0.89      0.89       413
weighted avg       0.89      0.89      0.89       413
```

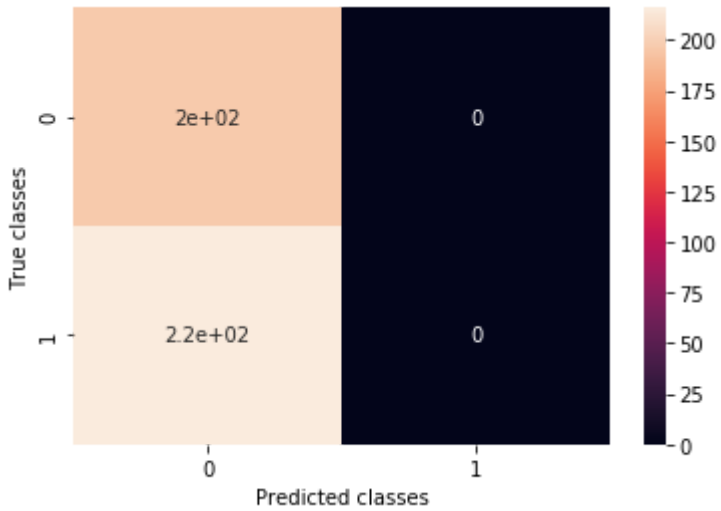



# Classifieur SVC & Sa matrice de confusion

In [28]:

```
clf = SVC(gamma='auto')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('F1 score(Accuracy): ', f1_score(y_test, y_pred, average='micro'))

## classification report
conf = confusion_matrix(y_test, y_pred)
print ('\n matrice de confusion \n',conf)

print ('\n',classification_report(y_test,
                                  y_pred))

## classification chart
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True)
plot.ylabel('True classes')
plot.xlabel('Predicted classes')
plot.show()
```

```
F1 score(Accuracy):  0.47699757869249393

 matrice de confusion
 [[197   0]
 [216   0]]

              precision    recall  f1-score   support

           1       0.48      1.00      0.65       197
           3       0.00      0.00      0.00       216

    accuracy                           0.48       413
   macro avg       0.24      0.50      0.32       413
weighted avg       0.23      0.48      0.31       413


/home/karim/env/lib/python3.6/site-packages/sklearn/metrics/_classific
ation.py:1272: UndefinedMetricWarning: Precision and F-score are ill-d
efined and being set to 0.0 in labels with no predicted samples. Use `
zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



## Cross validation (K_Fold)

## Mise en place de cross validation ( Evalutaion de notre modèle) avec le classifieur Gaussian naive bayes

Pour que notre modèle soit appris sur plusieurs jeux de données, on a appliqué 10-fold cross validation pour évaluer la qualité du modèle. En effet, Le jeu de données sera découpé en 10 partie. Du coup, notre modèle sera entrainé sur 9 partie et testé sur une.

In [29]:

```
seed=7
k_fold = KFold(n_splits=10, shuffle=True, random_state=seed)
clf = GaussianNB()

scoring = 'accuracy'
t0 = time()
score = cross_val_score(clf, X, y, cv=k_fold, scoring=scoring)
print("Réalisé en %0.3fs" % (time() - t0))

print('Les différentes accuracy pour les 10 évaluations sont : \n',
      score,'\n')
print ('Accuracy moyenne : ',score.mean(),
       ' standard deviation', score.std())
```

```
Réalisé en 3.608s
Les différentes accuracy pour les 10 évaluations sont :
 [0.93478261 0.87681159 0.89130435 0.91304348 0.91970803 0.94160584
 0.88321168 0.9270073  0.86131387 0.89781022]

Accuracy moyenne :  0.9046598963292076  standard deviation 0.025314498
84437806
```

## Test de plusieurs classifieurs à la fois Pour trouver le meilleur

- KNeighborsClassifier
- DecisionTreeClassifier
- GaussianNB
- SVC et RandomForestClassifier

## Test du modèle pour chaque classifieur

In [30]:

```
seed = 7
scoring = 'accuracy'
models = []
# Ajout des classifieurs
models.append(('NB', GaussianNB()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DTree', DecisionTreeClassifier()))
models.append(('LinearSVC', LinearSVC()))
models.append(('RFO', RandomForestClassifier()))
models.append(('SVM', SVC(gamma='auto')))

results = []
names = []
for name,model in models:
    kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
    t0 = time()
    cv_results = cross_val_score(model, X, y, cv=kfold, scoring=scoring)

    print("Réalisé en %0.3fs" % (time() - t0))
    print('Les différentes accuracy pour les 10 évaluations sont : \n',
      cv_results,'\n')

    results.append(cv_results)
    names.append(name)
    print ('Accuracy moyenne de : ', name," = ", cv_results.mean(), ' standard devi
    #print(msg)
```

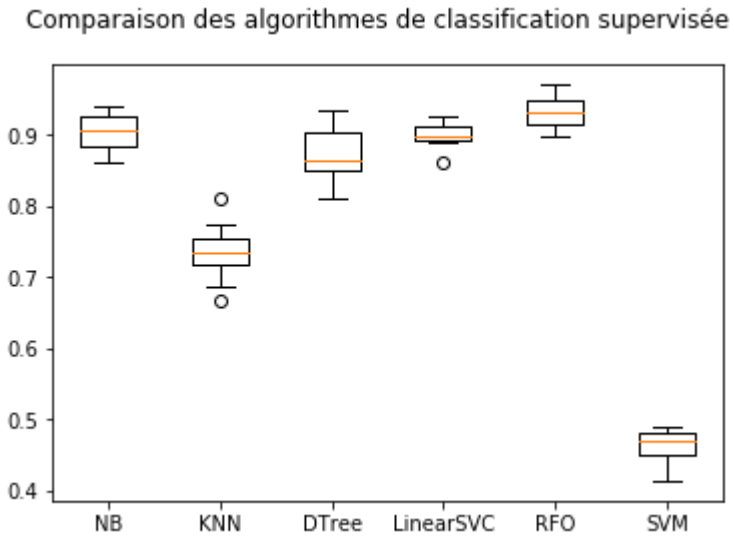
```
Réalisé en 3.609s
Les différentes accuracy pour les 10 évaluations sont :
 [0.93478261 0.87681159 0.89130435 0.91304348 0.91970803 0.94160584
 0.88321168 0.9270073  0.86131387 0.89781022]

Accuracy moyenne de :  NB  =  0.9046598963292076  standard deviation
0.02531449884437806
Réalisé en 90.976s
Les différentes accuracy pour les 10 évaluations sont :
 [0.66666667 0.75362319 0.8115942  0.7173913  0.75182482 0.72262774
 0.72262774 0.68613139 0.74452555 0.77372263]

Accuracy moyenne de :  KNN  =  0.7350735216333439  standard deviatio
n 0.03969948510583537
Réalisé en 28.931s
Les différentes accuracy pour les 10 évaluations sont :
 [0.86231884 0.84057971 0.84782609 0.86231884 0.91240876 0.93430657
 0.90510949 0.81021898 0.86861314 0.90510949]

Accuracy moyenne de :  DTree  =  0.8748809901618534  standard deviat
ion 0.03634814955774586
Réalisé en 2.810s
Les différentes accuracy pour les 10 évaluations sont :
 [0.89855072 0.91304348 0.89130435 0.89855072 0.89051095 0.9270073
 0.91240876 0.89781022 0.86131387 0.9270073 ]

Accuracy moyenne de :  LinearSVC  =  0.9017507669522903  standard de
viation 0.01850274402574869
```

```
Réalisé en 69.442s
Les différentes accuracy pour les 10 évaluations sont :
 [0.94927536 0.92028986 0.91304348 0.92028986 0.94890511 0.97080292
 0.95620438 0.91240876 0.89781022 0.94160584]

Accuracy moyenne de :  RFO  =  0.933063577700201  standard deviation
0.022232998366967956
Réalisé en 487.623s
Les différentes accuracy pour les 10 évaluations sont :
 [0.44927536 0.44927536 0.47101449 0.41304348 0.48905109 0.46715328
 0.45255474 0.48175182 0.47445255 0.48175182]

Accuracy moyenne de :  SVM  =  0.46293240241193273  standard deviati
on 0.021391604384952256
```

**Afficher les résultats des différents classifieurs(boxplot)**

In [31]:

```
fig = plot.figure()
fig.suptitle('Comparaison des algorithmes de classification supervisée')
ax = fig.add_subplot(111)
plot.boxplot(results)
ax.set_xticklabels(names)
plot.show()
```




# Hyperparameters and Visualization

# 1- Naive Bayes

In [32]:

```
#from sklearn.preprocessing import PowerTransformer

nb_classifier = GaussianNB()

params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}

gs_NB = GridSearchCV(estimator=nb_classifier,
                     param_grid=params_NB,
                     cv=5,
                     verbose=1,
                     scoring='accuracy')

#Data_transformed = PowerTransformer().fit_transform(Data)

gs_NB.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:  2.5min finished
```

Chercher la meilleure accuracy pour les paramètres

In [33]:

```
gs_NB.best_params_

print ('meilleur score ',
       gs_NB.best_score_,'\n')
print ('meilleurs paramètres',
       gs_NB.best_params_,'\n')
print ('meilleur estimateur',       gs_NB.best_estimator_,'\n')
```

```
meilleur score  0.8366472366148532

meilleurs paramètres {'var_smoothing': 1.232846739442066e-06}

meilleur estimateur GaussianNB(priors=None, var_smoothing=1.2328467394
42066e-06)
```

***Visualisation de résultats de GaussinNB grid search***

In [34]:

```
results_NB = pd.DataFrame(gs_NB.cv_results_['params'])
results_NB['test_score'] = gs_NB.cv_results_['mean_test_score']

alt.Chart(results_NB,
          title='Compraison de performance de  GaussinNB'
         ).mark_line(point=True).encode(
    alt.X('var_smoothing', title='Var. Smoothing'),
    alt.Y('test_score', title='Mean CV Score', scale=alt.Scale(zero=False))
).interactive()
```

Out[34]:

## 2- Random Forest

In [35]:

```
params_RFO = {'n_estimators': [4, 6, 9],
              'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1,5,8]
             }

gs_RFO = GridSearchCV(estimator=RandomForestClassifier(),
                      param_grid=params_RFO,
                      scoring='accuracy',
                      cv=5,
                      n_jobs=-1,
                     iid=True,
                     return_train_score=True)
```

In [36]:

```
gs_RFO.fit(X_train, y_train)
print ('meilleur score ', gs_RFO.best_score_,'\n')
print ('meilleurs paramètres', gs_RFO.best_params_,'\n')
print ('meilleur estimateur', gs_RFO.best_estimator_,'\n')
```

```
meilleur score  0.7148803329864725

meilleurs paramètres {'criterion': 'gini', 'max_depth': 10, 'max_featu
res': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estima
tors': 9}

meilleur estimateur RandomForestClassifier(bootstrap=True, ccp_alpha=
0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features='a
uto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=9,
                       n_jobs=None, oob_score=False, random_state=Non
e,
                       verbose=0, warm_start=False)
```

In [37]:

```
results_RFO = pd.DataFrame(gs_RFO.cv_results_['params'])
results_RFO['test_score'] = gs_RFO.cv_results_['mean_test_score']
results_RFO.columns

alt.Chart(results_RFO,
          title='Comparaison de performance de RandomForest'
         ).mark_line(point=True).encode(
    alt.X('max_depth', title='Maximum Depth'),
    alt.Y('test_score', title='Mean CV Score', aggregate='average', scale=alt.Scale
    color='criterion'
).interactive()
```

Out[37]:

A partir de l'histogramme nous pouvons remarquer que le meilleur hyperparamètre est le suivant: entropy avec une profondeur maximale de 10 et une valeur min_samples_split de 2.

# 3- Decision Tree

In [38]:

```
df_classifier = DecisionTreeClassifier(random_state=999)

params_DT = {'criterion': ['gini', 'entropy'],
             'max_depth': [1, 2, 3, 4, 5, 6, 7, 8],
             'min_samples_split': [2, 3]}

gs_DT = GridSearchCV(estimator=df_classifier,
                     param_grid=params_DT,
                     cv=5,
                     verbose=1,
                     scoring='accuracy')

gs_DT.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.
[Parallel(n_jobs=1)]: Done 160 out of 160 | elapsed:  1.6min finished
```

In [39]:

```
gs_DT.best_params_
```

Out[39]:

```
{'criterion': 'entropy', 'max_depth': 8, 'min_samples_split': 2}
```

In [40]:

```

print ('meilleur score ', gs_DT.best_score_,'\n')
print ('meilleurs paramètres ', gs_DT.best_params_,'\n')
print ('meilleur estimateur ', gs_DT.best_estimator_,'\n')
```

```
meilleur score  0.7200669257340242

meilleurs paramètres  {'criterion': 'entropy', 'max_depth': 8, 'min_sa
mples_split': 2}

meilleur estimateur  DecisionTreeClassifier(ccp_alpha=0.0, class_weigh
t=None, criterion='entropy',
                       max_depth=8, max_features=None, max_leaf_nodes=
None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecat
ed',
                       random_state=999, splitter='best')
```

Nous observons que le meilleur ensemble d'hyperparamètres est le suivant: entropy avec une profondeur maximale de 8 et une valeur min_samples_split de 2.

In [41]:

```
results_DT = pd.DataFrame(gs_DT.cv_results_['params'])
results_DT['test_score'] = gs_DT.cv_results_['mean_test_score']
results_DT.columns

alt.Chart(results_DT,
          title='Performance  de DecisionTree'
         ).mark_line(interpolate="basis").encode(
    alt.X('max_depth', title='Maximum Depth'),
    alt.Y('test_score', title='Mean CV Score', aggregate='average', scale=alt.Scale
    color='criterion'
).interactive()
```

Out[41]:

## Mise en place d'un gridsearch pour trouver le meilleur classifieur

In [42]:

```
params_RFO = {'n_estimators': [4, 6, 9],
              'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1,5,8]
             }
gs_RFO = GridSearchCV(estimator=RandomForestClassifier(),
                      param_grid=params_RFO,
                      scoring='accuracy',
                      cv=5,
                      n_jobs=-1,
                     iid=True,
                     return_train_score=True)



params_DT = {'criterion': ['gini', 'entropy'],
             'max_depth': [1, 2, 3, 4, 5, 6, 7, 8],
             'min_samples_split': [2, 3]}
gs_DT = GridSearchCV(estimator=DecisionTreeClassifier(random_state=999),
                      param_grid=params_DT,
                      cv=5,
                      verbose=1,
                      scoring='accuracy')


params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
gs_NB = GridSearchCV(estimator=GaussianNB(),
                      param_grid=params_NB,
                      cv=5,
                      verbose=1,
                      scoring='accuracy')


grids = [gs_DT, gs_RFO, gs_NB]

grid_dict={0:'Decision Tree', 1:'Random Forest', 2:'Naive Bayes'}

best_acc = 0.0
best_clf = 0.0
best_gs = ''

for idx,gs in enumerate(grids):
    print('\nClassifier: %s' % grid_dict[idx])
    gs.fit(X_train, y_train)
    print('Meilleurs paramètres : %s' % gs.best_params_)
    print ("meilleur score d'accurary : ",gs.best_score_,'\n')
    print ('meilleur estimateur : ',gs.best_estimator_,'\n')


    # Prediction sur le jeu de test avec les meilleurs paramètres
    t0 = time()
    result = gs.predict(X_test)
    print("Score d'accuracy pour les meilleurs paramètres sur jeu de test : %.3f"
    print ('\n matrice de confusion \n',confusion_matrix(y_test, result))
    print ('\n',classification_report(y_test, result))

    if accuracy_score(y_test, result) > best_acc:
```

```
        best_acc = accuracy_score(y_test, result)
        best_gs = gs
        best_clf = idx


print('\nClassifier avec la meilleur accuracy sur le jeu de test\n', grid_dict[best
```

```
Classifier: Decision Tree
Fitting 5 folds for each of 32 candidates, totalling 160 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.
[Parallel(n_jobs=1)]: Done 160 out of 160 | elapsed:  1.6min finished

Meilleurs paramètres : {'criterion': 'entropy', 'max_depth': 8, 'min_s
amples_split': 2}
meilleur score d'accurary :  0.7200669257340242

meilleur estimateur :  DecisionTreeClassifier(ccp_alpha=0.0, class_wei
ght=None, criterion='entropy',
                       max_depth=8, max_features=None, max_leaf_nodes=
None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecat
ed',
                       random_state=999, splitter='best')

Score d'accuracy pour les meilleurs paramètres sur jeu de test : 0.746

 matrice de confusion
 [[151  46]
 [ 59 157]]

              precision    recall  f1-score   support

           1       0.72      0.77      0.74       197
           3       0.77      0.73      0.75       216

    accuracy                           0.75       413
   macro avg       0.75      0.75      0.75       413
weighted avg       0.75      0.75      0.75       413


Classifier: Random Forest
Meilleurs paramètres : {'criterion': 'entropy', 'max_depth': 10, 'max_
features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_e
stimators': 6}
meilleur score d'accurary :  0.7200832466181062

meilleur estimateur :  RandomForestClassifier(bootstrap=True, ccp_alph
a=0.0, class_weight=None,
                       criterion='entropy', max_depth=10, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
```

```
                       min_weight_fraction_leaf=0.0, n_estimators=6,
                       n_jobs=None, oob_score=False, random_state=Non
e,
                       verbose=0, warm_start=False)

 Score d'accuracy pour les meilleurs paramètres sur jeu de test : 0.705

  matrice de confusion
  [[172  25]
  [ 97 119]]

               precision    recall  f1-score   support

           1       0.64      0.87      0.74       197
           3       0.83      0.55      0.66       216

    accuracy                           0.70       413
   macro avg       0.73      0.71      0.70       413
weighted avg       0.74      0.70      0.70       413


 Classifier: Naive Bayes
 Fitting 5 folds for each of 100 candidates, totalling 500 fits

 [Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
 t workers.
 [Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:  2.3min finished

 Meilleurs paramètres : {'var_smoothing': 1.232846739442066e-06}
 meilleur score d'accurary :  0.8366472366148532

 meilleur estimateur :  GaussianNB(priors=None, var_smoothing=1.2328467
 39442066e-06)

 Score d'accuracy pour les meilleurs paramètres sur jeu de test : 0.860

  matrice de confusion
  [[156  41]
  [ 17 199]]

               precision    recall  f1-score   support

           1       0.90      0.79      0.84       197
           3       0.83      0.92      0.87       216

    accuracy                           0.86       413
   macro avg       0.87      0.86      0.86       413
weighted avg       0.86      0.86      0.86       413


 Classifier avec la meilleur accuracy sur le jeu de test
  Naive Bayes
```

In [43]:

```
classifiers = {
    'RandomForestClassifier': RandomForestClassifier(),
    'DecisionTreeClassifier': DecisionTreeClassifier(),
    'NaiveBayesClassifier': GaussianNB()
}

params = {'RandomForestClassifier' :
          [{'n_estimators': [4, 6, 9]},
            {'max_features': ['log2', 'sqrt','auto']},
            {'criterion': ['entropy', 'gini']},
            {'max_depth': [2, 3, 5, 10]},
            {'min_samples_split': [2, 3, 5]},
            {'min_samples_leaf': [1,5,8]}],
           'DecisionTreeClassifier':
            [{'max_depth': [1,2,3,4,5,6,7,8,9,10]},
            {'criterion': ['gini', 'entropy']},
            {'min_samples_leaf': [1,2,3,4,5,6,7,8,9,10]}],
          'NaiveBayesClassifier':
            [{'var_smoothing': np.logspace(0,-9, num=100)}]
         }

class Result:
     def __init__(self,name, score, parameters):
         self.name = name
         self.score = score
         self.parameters = parameters
     def __repr__(self):
         return repr((self.name, self.score, self.parameters))


results = []
for key,value in classifiers.items():
    gd_sr = GridSearchCV(estimator=value,
                         param_grid=params[key],
                         scoring='accuracy',
                         cv=10,
                         n_jobs=1,
                         iid=True)
    gd_sr.fit(X_train, y_train)
    result=Result(key,gd_sr.best_score_,
                  gd_sr.best_estimator_)
    results.append(result)



results=sorted(results,
               key=lambda result: result.score,
               reverse=True)

print ('Le meilleur resultat : \n')
print ('Classifier : ',results[0].name,
       ' score %0.4f' %results[0].score,
       ' avec ',results[0].parameters,'\n')

print ('Tous les résultats : \n')
for result in results:
    print ('Classifier : ',result.name,
           ' score %0.4f' %result.score,
           ' avec ',result.parameters,'\n')
```

```
Le meilleur resultat :

Classifier :  RandomForestClassifier  score 0.9126  avec  RandomFore
stClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_feature
s='log2',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split
=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10
0,
                       n_jobs=None, oob_score=False, random_state=No
ne,
                       verbose=0, warm_start=False)

Tous les résultats :

Classifier :  RandomForestClassifier  score 0.9126  avec  RandomFore
stClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
```

## Sauvegarde le pipeline

In [44]:

```
pipeline = Pipeline([
    ('scaler',StandardScaler()),
    ('clf', GaussianNB())
])


t0 = time()
print ("Lancement du fit \n")
pipeline.fit(X_train, y_train)
print("Fit réalisé en %0.3fs" % (time() - t0))

result = pipeline.predict(X_test)
print('\n accuracy:',accuracy_score(result, y_test),'\n')

conf = confusion_matrix(y_test, result)
print ('\n matrice de confusion \n',conf)



print ('\n',classification_report(y_test, result))

print("\nSauvegarde du pipeline grid search avec succès")
filename = 'bestModel.pkl'
pickle.dump(pipeline, open(filename, 'wb'))
```

```
Lancement du fit

Fit réalisé en 0.724s

 accuracy: 0.8934624697336562


 matrice de confusion
 [[172  25]
 [ 19 197]]

              precision    recall  f1-score   support

           1       0.90      0.87      0.89       197
           3       0.89      0.91      0.90       216

    accuracy                           0.89       413
   macro avg       0.89      0.89      0.89       413
weighted avg       0.89      0.89      0.89       413


Sauvegarde du pipeline grid search avec succès
```

In [45]:

```
print ("Chargement du modèle \n")
filename = 'bestModel.pkl'
clf_loaded = pickle.load(open(filename, 'rb'))


print ("Sélection aléatoire de 15 assertions \n")
samples=[]
samples_result=[]
sample_new=[]
for i in df['truthRating'].head(15).index:
    sample_new.append(i)
    samples.append(X.values[i])
    samples_result.append(df.loc[ i , 'truthRating'])

print ("Prédiction des assertions séléctionnées\n")

result = clf_loaded.predict(samples)

print ("Valeurs réelles vs. valeurs prédites\n")
for i in range(len(result)):
    print ("assertion : ",sample_new[i],
           " \t réelle ", samples_result[i],
           "\t prédite ", result [i])
```

```
Prédiction des assertions séléctionnées

Valeurs réelles vs. valeurs prédites

assertion :  4          réelle  1      prédite  1
assertion :  7          réelle  1      prédite  1
assertion :  9          réelle  1      prédite  1
assertion :  10         réelle  3      prédite  1
assertion :  11         réelle  1      prédite  1
assertion :  19         réelle  1      prédite  1
assertion :  21         réelle  1      prédite  1
assertion :  22         réelle  1      prédite  1
assertion :  24         réelle  1      prédite  1
assertion :  25         réelle  1      prédite  1
assertion :  26         réelle  1      prédite  1
assertion :  27         réelle  1      prédite  1
assertion :  28         réelle  1      prédite  1
assertion :  30         réelle  1      prédite  1
assertion :  31         réelle  3      prédite  3
```

# Partie 2: Dataset {VRAI et FAUX} vs. {MIXTURE})

In [56]:

```
df = df_part2[(df['truthRating'] == 3) | (df_part2['truthRating'] == 1) | (df_part2
df.loc[df.truthRating==1, 'truthRating'] = 7
df.loc[df.truthRating==3, 'truthRating'] = 7
```

In [58]:

```
# enlever les valeurs manquantes
df.dropna(inplace=True)
array = df.values
X = array[:,  [0,1,2,3,4]] # variable d'apprentissage
y = array[:, 5]
y=y.astype('int')          # variable de prédiction

# OverSampling
ros = RandomOverSampler(random_state=42)
X_res, y_res = ros.fit_resample(X, y)
X = X_res
y = y_res

# prétraitements des données
for i in range(X.shape[0]):
    X[i] = clean_assertions(X[i])

# Mise en place d'un pipeline des données et appliquer TfidfVectorizer (pour transf
df2 = pd.DataFrame(data=X, columns=["text", "headline", "source", "keywords", "auth
transformer = FeatureUnion([
                ('text_tfidf',
                  Pipeline([('extract_field', FunctionTransformer(lambda x: x['text
                            ('tfidf', TfidfVectorizer(ngram_range=(1, 2)))])),
                ('headline_tfidf',
                  Pipeline([('extract_field', FunctionTransformer(lambda x:  x["hea
                            ('tfidf', TfidfVectorizer(ngram_range=(1, 2)))])),
                ('keywords_tfidf',
                  Pipeline([('extract_field', FunctionTransformer(lambda x:  x["key
                            ('tfidf', TfidfVectorizer(ngram_range=(1, 2)))])),
                ('author_tfidf',
                  Pipeline([('extract_field', FunctionTransformer(lambda x:  x["aut
                            ('tfidf', TfidfVectorizer(ngram_range=(1, 2)))])),
                ('source_tfidf',
                  Pipeline([('extract_field',
                              FunctionTransformer(lambda x:  x["source"], validate=
                            ('tfidf', TfidfVectorizer())])),
                ])

transformer.fit(df2)
text_vocab = transformer.transformer_list[0][1].steps[1][1].get_feature_names()
headline_vocab = transformer.transformer_list[1][1].steps[1][1].get_feature_names()
keywords_vocab = transformer.transformer_list[2][1].steps[1][1].get_feature_names()
author_vocab = transformer.transformer_list[3][1].steps[1][1].get_feature_names()
source_vocab = transformer.transformer_list[4][1].steps[1][1].get_feature_names()
vocab = text_vocab + headline_vocab + keywords_vocab + author_vocab  + source_vocab

X = pd.DataFrame(
    data=transformer.transform(df2).toarray(),
    columns=vocab)


# créer le jeu de données d'apprentissage(70%) et de test(30%)
validation_size=0.69
testsize= 1-validation_size
seed=30
X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=seed,
```

```
                                                    test_size=testsize)


# cross validation
seed=7
k_fold = KFold(n_splits=10, shuffle=True, random_state=seed)
```

## Accuracy moyenne de différents classifieurs

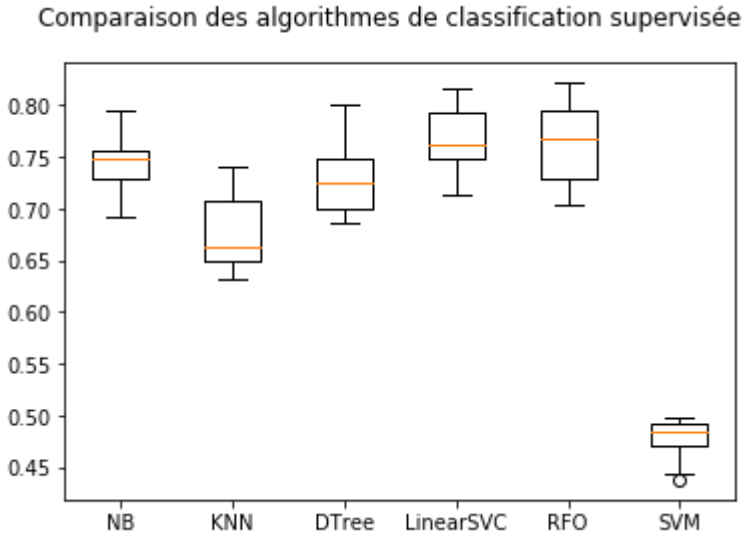
In [59]:

```
seed = 7
scoring = 'accuracy'
models = []
models.append(('NB', GaussianNB()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DTree', DecisionTreeClassifier()))
models.append(('LinearSVC', LinearSVC()))
models.append(('RFO', RandomForestClassifier()))
models.append(('SVM', SVC(gamma='auto')))
results = []
names = []
for name,model in models:
    kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
    t0 = time()
    cv_results = cross_val_score(model, X, y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    print (name,': accuracy moyenne = ', cv_results.mean(), ' standard deviation =
```

```
NB : accuracy moyenne =  0.7432432432432433  standard deviation =  0.0
29631286645763776
KNN : accuracy moyenne =  0.6783783783783784  standard deviation =  0.
03785713793490406
DTree : accuracy moyenne =  0.7281081081081082  standard deviation =
0.03367441757226492
LinearSVC : accuracy moyenne =  0.7686486486486487  standard deviation
=  0.03177716423070068
RFO : accuracy moyenne =  0.7643243243243243  standard deviation =  0.
03861745102592811
SVM : accuracy moyenne =  0.47729729729729736  standard deviation =
0.020232397051040976
```

In [60]:

```
# visualisaer les résultats de l'accuracy moyenne des classifieurs
fig = plot.figure()
fig.suptitle('Comparaison des algorithmes de classification supervisée')
ax = fig.add_subplot(111)
plot.boxplot(results)
ax.set_xticklabels(names)
plot.show()
```



In [61]:

```
# chercher la meilleure accuracy pour les paramètres
nb_classifier = GaussianNB()
params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
gs_NB = GridSearchCV(estimator=nb_classifier,
                     param_grid=params_NB,
                     cv=5,
                     verbose=1,
                     scoring='accuracy')
gs_NB.fit(X_train, y_train);
gs_NB.best_params_
print ('meilleur score ', gs_NB.best_score_,'\n')
print ('meilleurs paramètres', gs_NB.best_params_,'\n')
print ('meilleur estimateur', gs_NB.best_estimator_,'\n')

# visualiser les résultats de GaussinNB grid search
results_NB = pd.DataFrame(gs_NB.cv_results_['params'])
results_NB['test_score'] = gs_NB.cv_results_['mean_test_score']
alt.Chart(results_NB, title='Compraison de performance de  GaussinNB').mark_line(po
    alt.X('var_smoothing', title='Var. Smoothing'),
    alt.Y('test_score', title='Mean CV Score', scale=alt.Scale(zero=False))
).interactive()
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:  5.2min finished

meilleur score  0.7350949754901961

meilleurs paramètres {'var_smoothing': 0.006579332246575682}

meilleur estimateur GaussianNB(priors=None, var_smoothing=0.0065793322
46575682)
```

Out[61]:

## Mise en place d'un gridsearch pour trouver le meilleur classifieur

In [62]:

```
params_RFO = {'n_estimators': [4, 6, 9],
              'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1,5,8]
             }
gs_RFO = GridSearchCV(estimator=RandomForestClassifier(),
                      param_grid=params_RFO,
                      scoring='accuracy',
                      cv=5,
                      n_jobs=-1,
                     iid=True,
                     return_train_score=True)



params_DT = {'criterion': ['gini', 'entropy'],
             'max_depth': [1, 2, 3, 4, 5, 6, 7, 8],
             'min_samples_split': [2, 3]}
gs_DT = GridSearchCV(estimator=DecisionTreeClassifier(random_state=999),
                      param_grid=params_DT,
                      cv=5,
                      verbose=1,
                      scoring='accuracy')


params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
gs_NB = GridSearchCV(estimator=GaussianNB(),
                      param_grid=params_NB,
                      cv=5,
                      verbose=1,
                      scoring='accuracy')


grids = [gs_DT, gs_RFO, gs_NB]

grid_dict={0:'Decision Tree', 1:'Random Forest', 2:'Naive Bayes'}

best_acc = 0.0
best_clf = 0.0
best_gs = ''

for idx,gs in enumerate(grids):
    print('\nClassifier: %s' % grid_dict[idx])
    gs.fit(X_train, y_train)
    print('Meilleurs paramètres : %s' % gs.best_params_)
    print ("meilleur score d'accurary : ",gs.best_score_,'\n')
    print ('meilleur estimateur : ',gs.best_estimator_,'\n')


    # Prediction sur le jeu de test avec les meilleurs paramètres
    t0 = time()
    result = gs.predict(X_test)
    print("Score d'accuracy pour les meilleurs paramètres sur jeu de test : %.3f"
    print ('\n matrice de confusion \n',confusion_matrix(y_test, result))
    print ('\n',classification_report(y_test, result))

    if accuracy_score(y_test, result) > best_acc:
```

```
        best_acc = accuracy_score(y_test, result)
        best_gs = gs
        best_clf = idx


print('\nClassifier avec la meilleur accuracy sur le jeu de test\n', grid_dict[best
```

```
Classifier: Decision Tree
Fitting 5 folds for each of 32 candidates, totalling 160 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.
[Parallel(n_jobs=1)]: Done 160 out of 160 | elapsed:  3.5min finished

Meilleurs paramètres : {'criterion': 'entropy', 'max_depth': 8, 'min_s
amples_split': 2}
meilleur score d'accurary :  0.7233241421568627

meilleur estimateur :  DecisionTreeClassifier(ccp_alpha=0.0, class_wei
ght=None, criterion='entropy',
                       max_depth=8, max_features=None, max_leaf_nodes=
None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecat
ed',
                       random_state=999, splitter='best')

Score d'accuracy pour les meilleurs paramètres sur jeu de test : 0.702

 matrice de confusion
 [[225  47]
 [124 178]]

              precision    recall  f1-score   support

           2       0.64      0.83      0.72       272
           7       0.79      0.59      0.68       302

    accuracy                           0.70       574
   macro avg       0.72      0.71      0.70       574
weighted avg       0.72      0.70      0.70       574


Classifier: Random Forest
Meilleurs paramètres : {'criterion': 'gini', 'max_depth': 10, 'max_fea
tures': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_esti
mators': 6}
meilleur score d'accurary :  0.7100313479623824

meilleur estimateur :  RandomForestClassifier(bootstrap=True, ccp_alph
a=0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features='a
uto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
```

```
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=6,
                       n_jobs=None, oob_score=False, random_state=Non
e,
                       verbose=0, warm_start=False)

Score d'accuracy pour les meilleurs paramètres sur jeu de test : 0.693

 matrice de confusion
 [[226  46]
 [130 172]]

              precision    recall  f1-score   support

           2       0.63      0.83      0.72       272
           7       0.79      0.57      0.66       302

    accuracy                           0.69       574
   macro avg       0.71      0.70      0.69       574
weighted avg       0.72      0.69      0.69       574


Classifier: Naive Bayes
Fitting 5 folds for each of 100 candidates, totalling 500 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:  6.5min finished

Meilleurs paramètres : {'var_smoothing': 0.006579332246575682}
meilleur score d'accurary :  0.7350949754901961

meilleur estimateur :  GaussianNB(priors=None, var_smoothing=0.0065793
32246575682)

Score d'accuracy pour les meilleurs paramètres sur jeu de test : 0.739

 matrice de confusion
 [[224  48]
 [102 200]]

              precision    recall  f1-score   support

           2       0.69      0.82      0.75       272
           7       0.81      0.66      0.73       302

    accuracy                           0.74       574
   macro avg       0.75      0.74      0.74       574
weighted avg       0.75      0.74      0.74       574


Classifier avec la meilleur accuracy sur le jeu de test
 Naive Bayes
```

In [63]:

```
classifiers = {
    'RandomForestClassifier': RandomForestClassifier(),
    'DecisionTreeClassifier': DecisionTreeClassifier(),
    'NaiveBayesClassifier': GaussianNB()
}

params = {'RandomForestClassifier' :
          [{'n_estimators': [4, 6, 9]},
            {'max_features': ['log2', 'sqrt','auto']},
            {'criterion': ['entropy', 'gini']},
            {'max_depth': [2, 3, 5, 10]},
            {'min_samples_split': [2, 3, 5]},
            {'min_samples_leaf': [1,5,8]}],
           'DecisionTreeClassifier':
            [{'max_depth': [1,2,3,4,5,6,7,8,9,10]},
            {'criterion': ['gini', 'entropy']},
            {'min_samples_leaf': [1,2,3,4,5,6,7,8,9,10]}],
          'NaiveBayesClassifier':
            [{'var_smoothing': np.logspace(0,-9, num=100)}]
         }

class Result:
     def __init__(self,name, score, parameters):
         self.name = name
         self.score = score
         self.parameters = parameters
     def __repr__(self):
         return repr((self.name, self.score, self.parameters))


results = []
for key,value in classifiers.items():
    gd_sr = GridSearchCV(estimator=value,
                         param_grid=params[key],
                         scoring='accuracy',
                         cv=10,
                         n_jobs=1,
                         iid=True)
    gd_sr.fit(X_train, y_train)
    result=Result(key,gd_sr.best_score_,
                  gd_sr.best_estimator_)
    results.append(result)



results=sorted(results,
               key=lambda result: result.score,
               reverse=True)

print ('Le meilleur resultat : \n')
print ('Classifier : ',results[0].name,
       ' score %0.4f' %results[0].score,
       ' avec ',results[0].parameters,'\n')

print ('Tous les résultats : \n')
for result in results:
    print ('Classifier : ',result.name,
           ' score %0.4f' %result.score,
           ' avec ',result.parameters,'\n')
```

```
Le meilleur resultat :

Classifier :  RandomForestClassifier  score 0.7508  avec  RandomForest
Classifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=3,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=Non
e,
                       verbose=0, warm_start=False)

Tous les résultats :

Classifier :  RandomForestClassifier  score 0.7508  avec  RandomForest
Classifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=3,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=Non
e,
                       verbose=0, warm_start=False)

Classifier :  NaiveBayesClassifier  score 0.7422  avec  GaussianNB(pri
ors=None, var_smoothing=0.008111308307896872)

Classifier :  DecisionTreeClassifier  score 0.7249  avec  DecisionTree
Classifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=9, max_features=None, max_leaf_nodes=
None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecat
ed',
                       random_state=None, splitter='best')
```

In [66]:

```
pipeline = Pipeline([
    ('scaler',StandardScaler()),
    ('clf', RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False))
])


t0 = time()
print ("Lancement du fit \n")
pipeline.fit(X_train, y_train)
print("Fit réalisé en %0.3fs" % (time() - t0))

result = pipeline.predict(X_test)
print('\n accuracy:',accuracy_score(result, y_test),'\n')

conf = confusion_matrix(y_test, result)
print ('\n matrice de confusion \n',conf)



print ('\n',classification_report(y_test, result))

print("\n=========== Sauvegarde du pipeline grid search avec succès ============")
filename = 'bestModelMixture.pkl'
pickle.dump(pipeline, open(filename, 'wb'))
```

```
Lancement du fit

Fit réalisé en 2.510s

 accuracy: 0.7125435540069687


 matrice de confusion
 [[231  41]
 [124 178]]

              precision    recall  f1-score   support

           2       0.65      0.85      0.74       272
           7       0.81      0.59      0.68       302

    accuracy                           0.71       574
   macro avg       0.73      0.72      0.71       574
weighted avg       0.74      0.71      0.71       574


=========== Sauvegarde du pipeline grid search avec succès ===========
=
```